RACCOON
(configuRAble Compiler COmpiler frOnt-end generatioN)
User Manual

Ajax Compilers

January 23, 2013

Contact: info@ajaxcompilers.com
Website: http://www.ajaxcompilers.com/
Release Date: January 23, 2013
Version: 1.0.0
Rev. history:
v1.0.0 15-01-2013

1 Introduction

RACCOON (configuRAble Compiler COmpiler frOnt-end generatioN) is a tool for generating compiler front-ends using a BNF grammar. The current version of RACCOON produces C++ source code for a project reaching up to the construction of a parse tree. The output code uses the Flex tool for generating a lexical analyzer and the Bison tool for the generation of a LALR(1) parser. The tool automates the time consuming process of implementing the mechanism for parse tree generation and frees the user/developer from having to deal with it. This way the user can devote more time for subsequent compiler design tasks.

The input to RACCOON is a Bison BNF grammar. The grammar can be plain BNF rules and no actions or Bison switches are required to be provided by the user. Certain settings are provided as input as well, through a configuration file. These settings regard flex/bison tokens and nonterminals, class, variable, and file names in the output code. The output code consists of all the .cpp, .h, .l and .y files required to build a parse tree for the grammar the user provided as input. The only thing missing from the generated code is the regular expressions for the generated tokens. Some space is formed within the Flex (.l) file for providing the regular expressions for each token. Following the addition of the regular expressions by the user, the output code is ready for compilation.

RACCOON is available for Windows and Linux operating systems. Instructions are provided for setting up a C++ project with the output code in Visual Studio (Windows) and NetBeans (Linux). Visual C++ compiler is used in Windows and GCC (g++) in Linux. The generated code includes routines for the production of a Graphviz .dot file for the visualization of the parse tree.

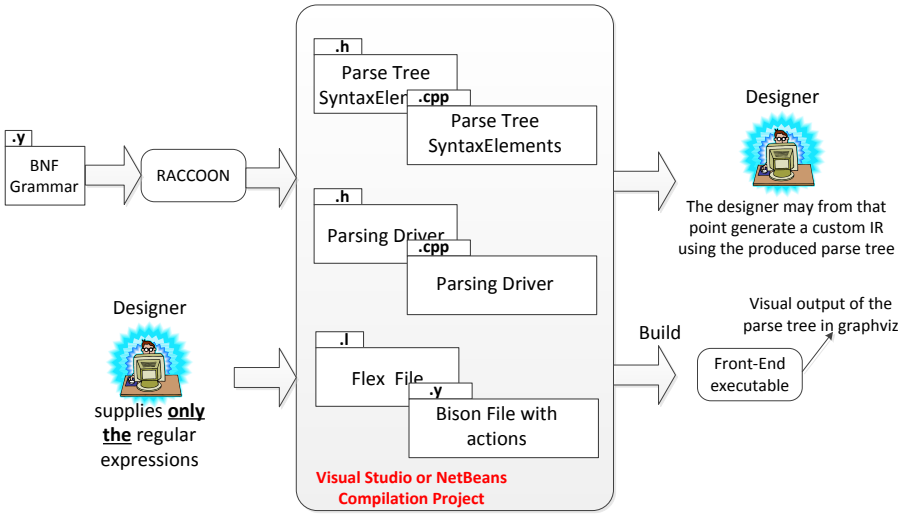

Figure 1: RACCOON methodology flow.

2 Contents

This RACCOON release has the following contents:

- *license.txt*. RACCOON license.
- *readme.pdf*. This file.
- *raccoon.exe* (Windows) or *raccoon* (Linux). The main executable file for RACCOON.
- *configuration.conf*. A sample configuration file.
- *demo.y*. A sample grammar that can be used for demonstration purposes.
- *demo_with_regex.l*. Flex file for getting the regular expressions for the sample grammar above.

3 Requirements

RACCOON is available for the following operating systems:

- Windows XP, Vista, 7, 8, Linux

Prerequisities for the compilation of the output code:

- Windows specific:
  - Visual Studio or another C++ IDE
  - GnuWin32 Flex package
    http://gnuwin32.sourceforge.net/packages/flex.htm
    http://gnuwin32.sourceforge.net/downlinks/flex.php
  - GnuWin32 Bison package
    http://gnuwin32.sourceforge.net/packages/bison.htm
    http://downloads.sourceforge.net/gnuwin32/bison-2.4.1-setup.exe
  - GnuWin32 M4 package
    http://gnuwin32.sourceforge.net/packages/m4.htm
    http://downloads.sourceforge.net/gnuwin32/m4-1.4.14-1-setup.exe
  - Flex and Bison Custom Build Rules (available only for Visual Studio 2005 and 2008)
    http://msdn.microsoft.com/en-us/library/aa730877(v=vs.80).aspx
    http://www.microsoft.com/en-us/download/details.aspx?id=14760
- Linux specific:
  - NetBeans or another C++ IDE
    http://netbeans.org/
  - Flex
  - Bison
  - M4
  - GCC (g++)
- Graphviz
  http://www.graphviz.org/

4 Installation

1. Obtain RACCOON distribution
   The latest version of RACCOON can be obtained at:
   http://www.ajaxcompilers.com/

2. Unpack
   Users need to unpack the distribution into a directory of their choice
   - For Windows use an utility like WinZip or WinRAR
   - For Linux use a similar utility like 7-Zip or unpack the file from the terminal
     $cd <directory_where_raccoon-vX.x.x.zip_exists >
     $apt-get install unzip
     $yum install unzip if you are Red Hat Linux/Fedora user
     $unzip raccoon-vX.x.x.zip

5 Usage

Users can run RACCOON in the following way:

$raccoon grammar_file configuration_file target_directory

- *grammar_file* is the path of the input Bison BNF grammar file.
- *configuration_file* provides the data used to configure RACCOON. Most of the configurations defined in this file regard naming conventions in the output code.
- *target_directory* is the directory where the tool generates the output files.

5.1 Configuration File

Before running the application, you have to create a configuration file. Here's a sample configuration file:

```
1 parser_class_name=DEMOParserClass
2 namespace_name=DEMOParser
3 driver_name=DEMOParser_driver
4 grammar_filename=DEMOParser
5 grammar_start_symbol=scdeclaration_unit
6 endoffile_token=ENDOFFILE
```

- *parser_class_name*. The name of the parser class generated by the Bison parser generator.
- *namespace_name*. The name of the namespace generated by the Bison parser generator containing, among others, the above class.
- *driver_name*. The name of the main class of the application. Every operation of the program is performed by calling functions of this class. It has functions that initiate operations like parsing the input file, constucting the parse tree, extracting the parse tree in a Graphviz .dot file and visualizing it.

- *grammar_filename*. The name of the .y and .l files.
- *grammar_start_symbol*. The name of the start symbol in the BNF grammar.
- *endoffile_token*. The name of the token specifying the end of file. Such a token must be declared in the grammar.

5.2 Input BNF Grammar File

The BNF grammar that RACCOON takes as input should be compliant with the syntax of Bison parser generator. No actions or Bison switches are required to be provided by the user. The grammar should include plain rules, nonterminals, tokens etc.

The Bison manual comes with the Bison installation. It is also available here:

http://www.gnu.org/software/bison/manual/index.html

6 Working with the Output Code

The code generated by RACCOON contains the following files. Please note that the parse tree is denoted as PT in the output code:

- PTDefines.h
- PTSyntaxElements.h
- Defines.h
- driver.h
- GFunctions.h
- PT2HLIRGen.cpp
- PTDefines.cpp
- PTGraphEmmiter.cpp
- PTSyntaxElements.cpp
- Defines.cpp
- driver.cpp
- GFunctions.cpp
- *grammar_filename*.l
- *grammar_filename*.y

The *.y* file contains the Bison BNF grammar and the *.l* file contains Flex's input. Some space is formed within the Flex (*.l*) file for providing the regular expressions for each token.

PTSyntaxElements.h and *PTSyntaxElements.cpp* contain classes representing the syntax elements that comprise the parse tree. Some global declarations of mostly enums and string arrays assisting these files can be found in *PTDefines.h* and *PTDefines.cpp*. Other general declarations/definitions can be found in *Defines.h* and *Defines.cpp*.

Every syntax element class has a virtual function for emmiting graphviz code. The definitions of these functions can be found in *PTGraphEmmiter.cpp*.

Another virtual class is available that parses the tree and gives the user the ability to generate an Intermediate Representation (IR). Of course the user needs to provide the IR and develop the code for generating it within the provided functions. These functions' definitions are placed in *PT2HLIRGen.cpp*.

GFunctions.h and *GFunctions.cpp* contain functions that perform general tasks and can be used throughout the code. For example, they contain a custom strdup function (*StrDup_cpp*).

driver.h and *driver.cpp* contain the main class of the application described in the "configuration file" section. This class is responsible for conducting the whole process. It initiates each separate operation of the application.

There are two things missing from the output code to make it suitable for compilation. 1) The regular expressions in the *.l* file that must be provided by the user, and 2) a main function for the application that will create a driver object and start the procedure. The contents of a sample file containing the main function can be seen in the example bellow:

```
#include "driver.h"

void main(int argc, char **argv){
DEMOParser_driver driver(argc-1, &(argv[1]));
  driver.parse();
}
```

The locations where the regular expressions for each token should be placed within the *.l* file are marked with special strings. These strings are in the form:

regexp_*TOKEN_NAME*

If, for example, we have a token for the left parenthesis character named *LPARENTHESIS* in the grammar, the string *regexp_LPARENTHESIS* will be generated in the output *.l* file. This string can be replaced with the regular expression *"("*. Furthermore, the sequence of the regular expressions/tokens in the *.l* file may need rearrangement. For example a regular expression recognizing an identifier should be placed at the bottom of the file, so that it would be the last one in the sequence.

grammarCLEAN.y is a file that was not listed above with the rest. It is generated by RACCOON as well, but it is not needed for the compilation. This file contains a clean version of the output grammar where everything have been removed except from the tokens, nonterminals and rules that comprise the grammar. The user can be assisted by this file in case a requirement for the consultation of a pure BNF grammar comes up.

6.1 Set up on Windows/Visual Studio

In this section we explain the steps for setting up Visual Studio for compilation of the output code. Visual Studio 2005, 2008, 2010 and 2012 should all work for this. If you don't own a copy of Visual Studio you can download one of the freely available express editions:

- Microsoft Visual Studio 2008 Express Edition
  http://www.microsoft.com/en-us/download/details.aspx?id=6506
- Microsoft Visual Studio Express 2012 for Windows Desktop
  http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-windows-desktop

Visual Studio 2008 will be used in this guide. The file FlexBison.rules is used for automatically setting up Visual Studio Custom Build Steps for Flex and Bison. This file only works for Visual Studio 2005 and 2008. For Visual Studio 2010, 2012 you can find more information here: `http://msdn.microsoft.com/en-us/library/e85wte0k.aspx`

6.1.1 Flex/Bison Installation on Windows

1. Download GnuWin32 Flex, GnuWin32 Bison, GnuWin32 M4 and FlexBison.rules (Only for Visual Studio 2005, 2008).
2. Install all the above in "C:\GnuWin32\".
3. Add "C:\GnuWin32\bin" to the system *Path* variable.
   For Windows 7 go to the Control Panel and navigate to *System -> Advanced system settings -> Advanced -> Environment Variables...*.
   Find the *Path* variable in the System variables list and click *Edit...*.
   Type a semicolon at the end of the string followed by "C:\GnuWin32\bin".

6.1.2 Setting up Visual Studio 2008

After installing Flex and Bison, the following instructions will guide you through the configuration of Visual Studio for the compilation of RACCOON's output code.

1. Run Visual Studio
2. From the menu navigate to *File -> New -> Project -> Visual C++ -> Win32 Console Application*. Enter a name for the new project, browse a folder to save it and click *OK*. Select *Console application* and check the *Empty project* checkbox.
3. Copy all the RACCOON output files in the project folder.
4. Right click on the project name in the Solution Explorer and select *Add -> Existing Item...*.
5. Select all files: *grammar_filename.l*, *grammar_filename.y*, *Defines.cpp*, *Defines.h*, *driver.cpp*, *driver.h*, *GFunctions.cpp*, *GFunctions.h*, *PT2HLIRGen.cpp*, *PTDefines.cpp*, *PTDefines.h*, *PTGraphEmmiter.cpp*, *PTSyntaxElements.cpp*, *PTSyntaxElements.h*.

6. Right click on the project name in the Solution Explorer and select *Properties -> Configuration Properties -> C/C++*

7. Add "C:\GnuWin32\include" on *Additional Include Directories*.

8. Right click on the project name in the Solution Explorer and select *Custom Rules....*

9. Make sure *Flex and Bison Tools* is checked in the *Available Rule Files* list. If *Flex and Bison Tools* does not exist in this list click *Find Existing...*, navigate to the folder where you installed the FlexBison.rules file and select it.
   Click *OK* to close the *Visual C++ Custom Build Rule Files* window.

10. Right click on the project name in the Solution Explorer and select *Build*.

11. The compilation should fail. Right click on the project name in the Solution Explorer again and select *Add -> Existing Item....*

12. Select: *grammar_filename.tab.h*, *grammar_filename.tab.c*, *lex.grammar_filename.c*.

13. Right click on *grammar_filename.tab.c* from the Solution Explorer and select *Properties*.

14. From the *Property Pages* window go to *Configuration Properties -> C/C++ -> Advanced* and select *Compile as C++ Code (/TP)* from the *Compile As* dropdown. Click *OK* to apply the setting and close the window. Do the same for the file *lex.grammar_filename.c*.

15. Right click on the project name in the Solution Explorer and select *Add -> New Item...* and add a new *C++ File (.cpp)*.

16. Inside this new file, create a main function, include "driver.h" and create a driver object. The contents of a sample file containing the main function can be seen in the beginning of this section.

6.2 Set up on Linux/NetBeans

For the Linux version of RACCOON we used NetBeans to demonstrate the preparation of the output project for compilation. Ubuntu 12.04.1 LTS and NetBeans IDE 7.2 were used during the preparation of these instructions.

The applications listed in the Requirements section should be installed before starting the process. If you use Ubuntu you can install Flex, Bison and M4 through Ubuntu Software Center. Instructions on installing NetBeans can be found here: http://netbeans.org/ and GCC should already be installed in your system.

After successfully installing everything, the following instructions should be followed for the configuration of NetBeans for the compilation of RACCOON's output code.

1. Run NetBeans

2. From the menu navigate to *File -> New Project...*. Select *C/C++ -> C/C++ Application* and click *Next>*. Enter a name for the new project, browse a folder to save it, check the *Create Main File* checkbox, choose *C++* as the project type and *GNU Compiler Collection* as the *Tool Collection* and click *Finish*. A new project is now created and shown in the Projects list.

3. Copy all the RACCOON output files in the project folder.

4. Right click on the project name and select *Add Existing Item...*.

5. Select all files: *grammar_filename.l*, *grammar_filename.y*, *Defines.cpp*, *Defines.h*, *driver.cpp*, *driver.h*, *GFunctions.cpp*, *GFunctions.h*, *PT2HLIRGen.cpp*, *PTDefines.cpp*, *PTDefines.h*, *PTGraphEmmiter.cpp*, *PTSyntaxElements.cpp*, *PTSyntaxElements.h*.

6. Right click on the *.l* file and select *Properties*. On the *General* tab select *Custom Build Tool* on the *Tool* dropdown and click the *Apply* button. Now click the *Custom Build Step* tab, enter the following and click *OK* to save and close the *File Properties* window:
   Command Line:

flex -olex.*grammar_filename*.c -P*grammar_filename* -Cem *grammar_filename*.l

   Description:

Running Flex...

   Outputs:

lex.*grammar_filename*.c

7. Right click on the *.y* file and select *Properties*. On the *General tab* select *Custom Build Tool* on the *Tool* dropdown and click the *Apply* button. Now click the *Custom Build Step* tab, enter the following and click *OK* to save and close the *File Properties* window:
   Command Line:

bison -b *grammar_filename* -p *grammar_filename grammar_filename*.y

   Description:

Running bison...

   Outputs:

many_outputs.out

8. Right click on the *.l* file and select *Compile File*. Do the same for the *.y* file.

9. Right click on the project name and select *Add Existing Item....* Select: *grammar_filename.tab.h*, *grammar_filename.tab.c*, *lex.grammar_filename.c*.
10. Right click on *grammar_filename.tab.c* and select *Properties*. On the *General* tab select *C++ Compiler* on the *Tool* dropdown and click the *OK* button. Do the same for the file *lex.grammar_filename.c*.
11. Edit the main file. Include "driver.h" and create a driver object inside the main function. The contents of a sample file containing the main function can be seen in the beggining of this section.

6.3 Graphviz

Graphviz is an open source graph visualization software. It takes as input a text description of a graph in DOT language and creates visualized diagrams in various formats (GIF, PNG etc). The generated code uses Graphviz to visually present the parse tree to the user.

Note that the directory where the Graphviz bin files are installed needs to be included in the system *Path* variable.

7 Quick Tutorial

This is a small tutorial that will help the user get more familiar with the process.

1. Locate the file *demo.y* included in this RACCOON release. This file contains the grammar for a language similar to the syntax of C declarations.
2. Create a configuration file and execute RACCOON with *demo.y* and the configuration as input. The output code will be generated.
3. Install Flex/Bison and set them up according to the instructions given in this manual.
4. Install Graphviz as well.
5. Create a new VS/NetBeans project and set up the environment properly.
6. Create a main function for the project.
7. Fill in the missing regular expressions from the *.l* file.
   You can copy the contents of the *demo_with_regex.l* file (included in this release) inside your *.l* file. This file has the regular expressions included.
   Note that the regular expression for the token *IDENTIFIER* (*regexp_IDENTIFIER*) is rearranged and placed last in the sequence.
8. Compile the project.

8 Support

Please send your support requests to info@ajaxcompilers.com.